Never Stand Still

Classification (1)

COMP9417 Machine Learning & Data Mining

Term 1, 2021

Adapted from slides by Dr Michael Bain

Aims

This lecture will introduce you to machine learning approaches to the problem of classification. Following it you should be able to reproduce theoretical results, outline algorithmic techniques and describe practical applications for the topics:

- outline a framework for solving machine learning problems
- outline the general problem of induction
- describe issues of generalisation and evaluation for classification
- outline the use of a linear model as a 2-class classifier
- describe distance measures and how they are used in classification
  • outline the basic k-nearest neighbour classification method

Introduction

Classification (sometimes called *concept learning*) methods dominate machine learning . . .

. . . however, they often don't have convenient mathematical properties like regression, so are more complicated to analyse. The idea is to learn a *classifier*, which is usually a function mapping from an input data point to one of a set of discrete outputs, i.e., the classes.

We will mostly focus on classifier advantages and disadvantages as learning methods first and point to unifying ideas and approaches where applicable.

Classification

Example: Imagine that we want to automate the process of sorting incoming fish in a fish-packing plant. And as a pilot, we start by separating sea bass from salmon using some information collected through sensing.

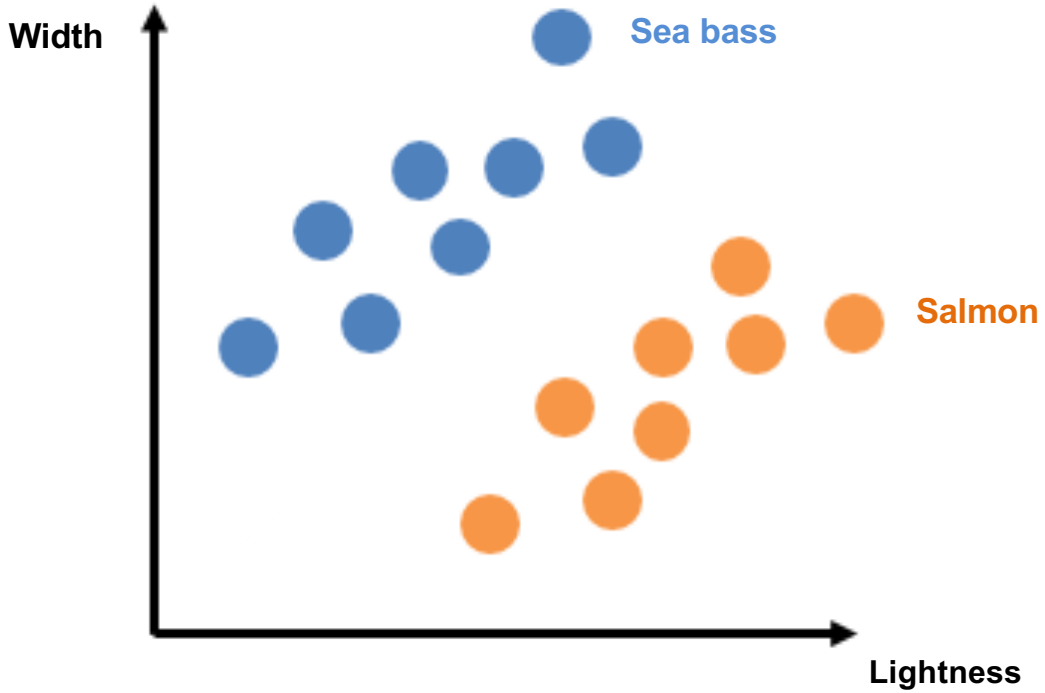
(a) (b)

Classification

Example: classifying sea bass vs. salmon

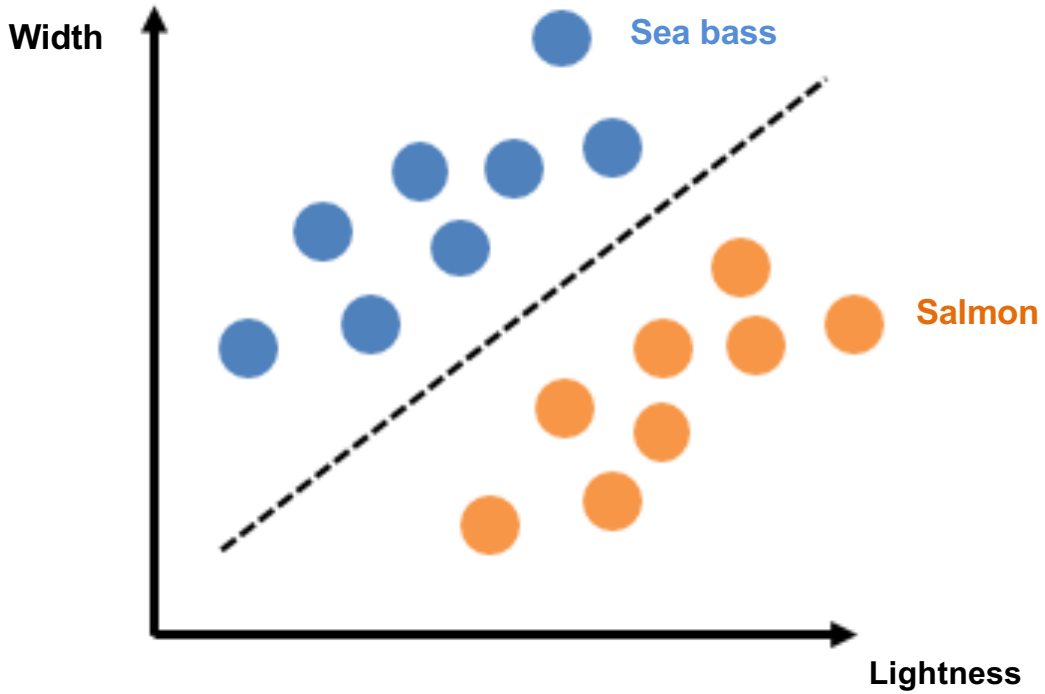
Features that can be used: width, length, weight, lightness, fins, eyes/mouth position, etc.

Question: how to separate these two classes?



Classification

Example: Maybe we can find a line that separates the two classes.



Classification

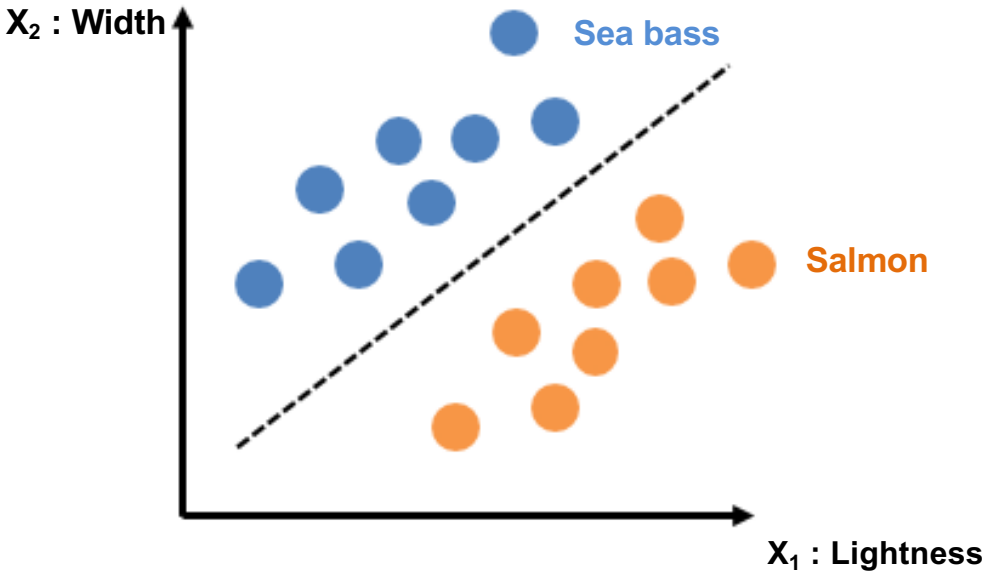
Example: If we find the line that separated the two classes, then how our algorithm makes prediction?

The line equation will look like:
$ax_1 + bx_2 + c = 0$

We can define a, b & c such that:

for any point above the line:
$ax_1 + bx_2 + c > 0$

and for any point below the line:
$ax_1 + bx_2 + c < 0$



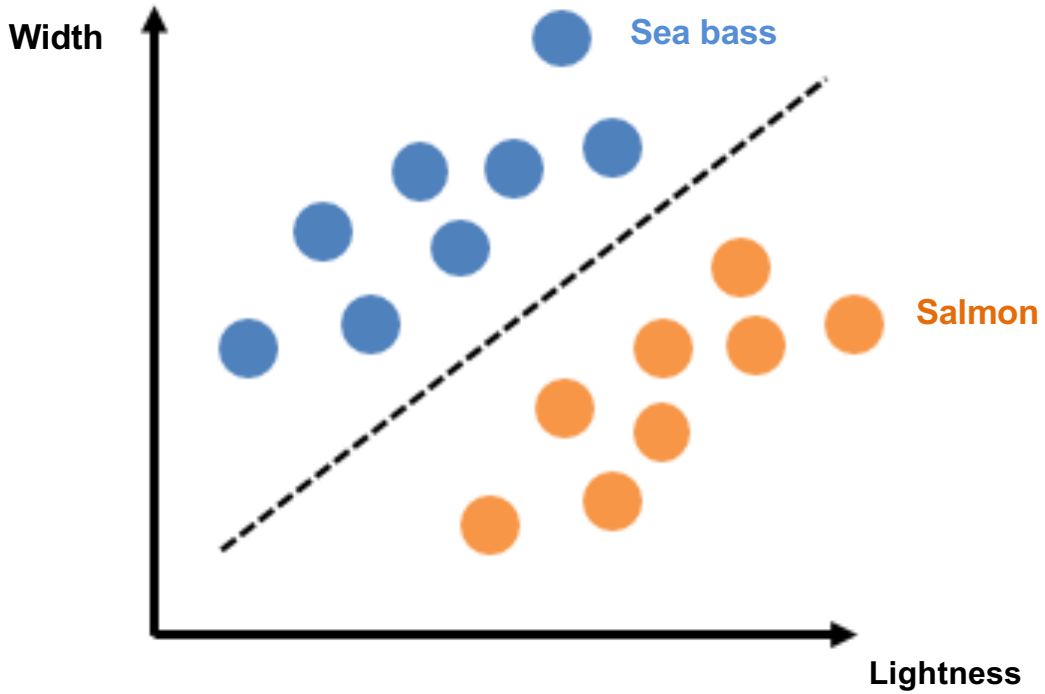
This type of classifier is called *linear classifier*. It is also a type of *discriminative learning* algorithm.

判别

Classification

Example:

Can we do something different than finding the discriminative line (or some boundary) to be able to separate the two groups?

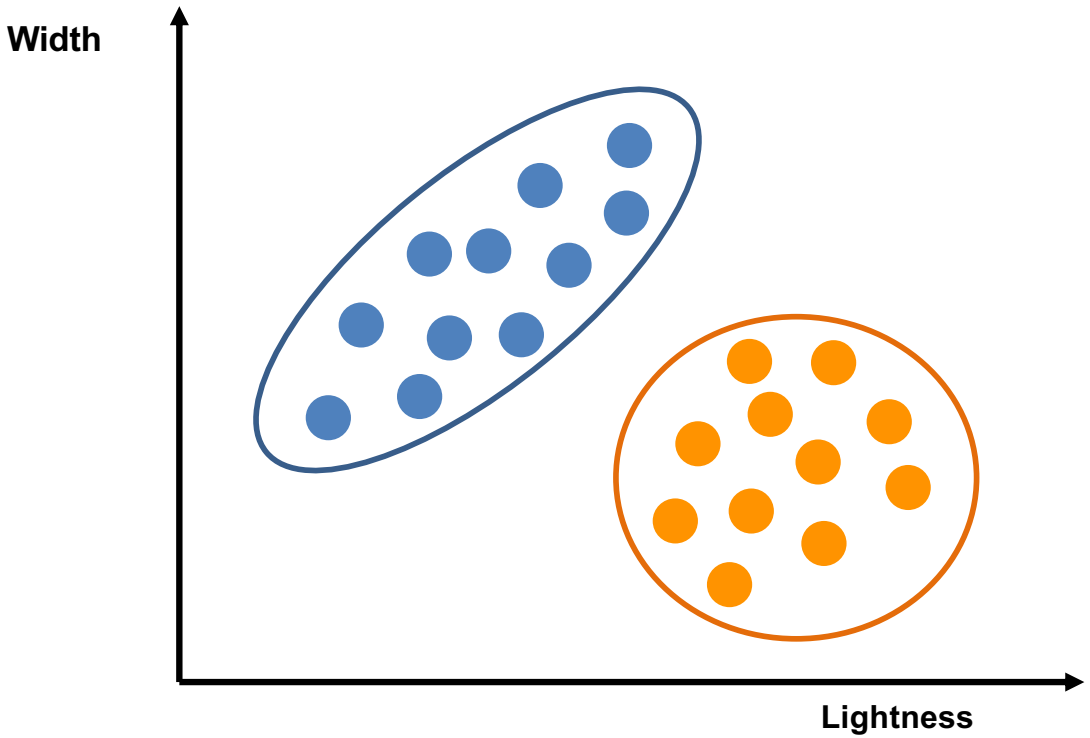


Classification

Example:

Instead of finding a discriminative line, maybe we can focus on one class at a time and build a model that describes how that class looks like; and then do the same for the other class. This type of models are called *generative learning algorithm*.

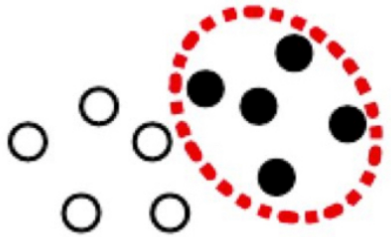
生成



Classification

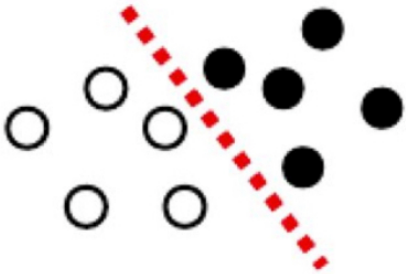
Generative algorithm: builds some models for each of the classes and then makes classification predictions based on looking at the test example and see it is more similar to which of the models.

- Learns $p(x|y)$ (and also $p(y)$, called class prior)
- So, we can get $p(x, y) = p(x|y)p(y)$
- It learns the mechanism by which the data has been generated



Discriminative algorithm: Do not build models for different classes, but rather focuses on finding a decision boundary that separates classes

- Learns $p(y|x)$



Classification

- To predict the output for sample x, in generative algorithm, we have to estimate $p(y|x)$:

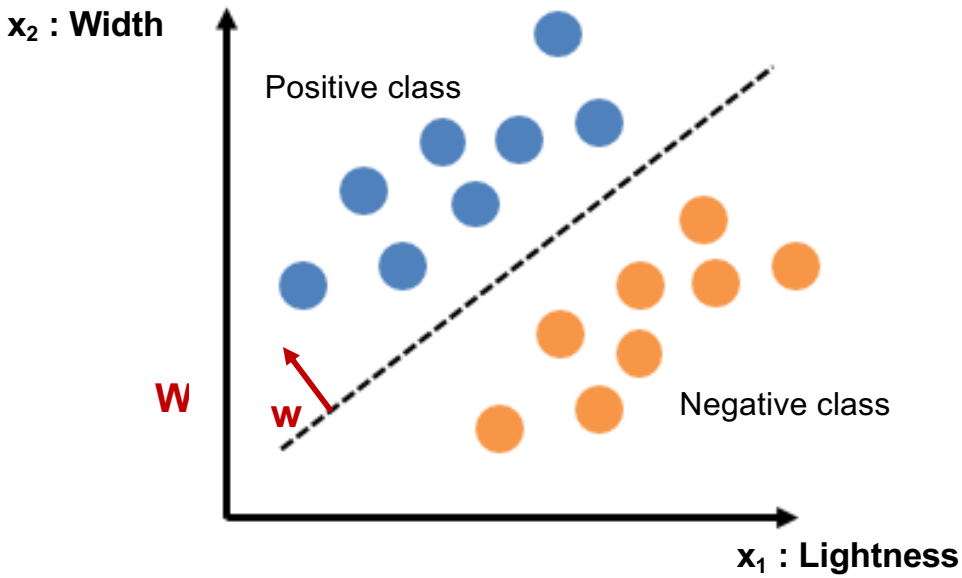
$$p(y=0|x) = \frac{p(x|y=0)p(y=0)}{p(x)}$$

$$p(y=1|x) = \frac{p(x|y=1)p(y=1)}{p(x)}$$

If $p(y=0|x) > p(y=1|x)$, then x belongs to class $y=0$ and otherwise to class $y=1$.

- For discriminative algorithm, we can directly have $p(y=0|x)$ and $p(y=1|x)$ and similar to above, if $p(y=0|x) > p(y=1|x)$, then x belongs to class $y=0$ and otherwise to class $y=1$.

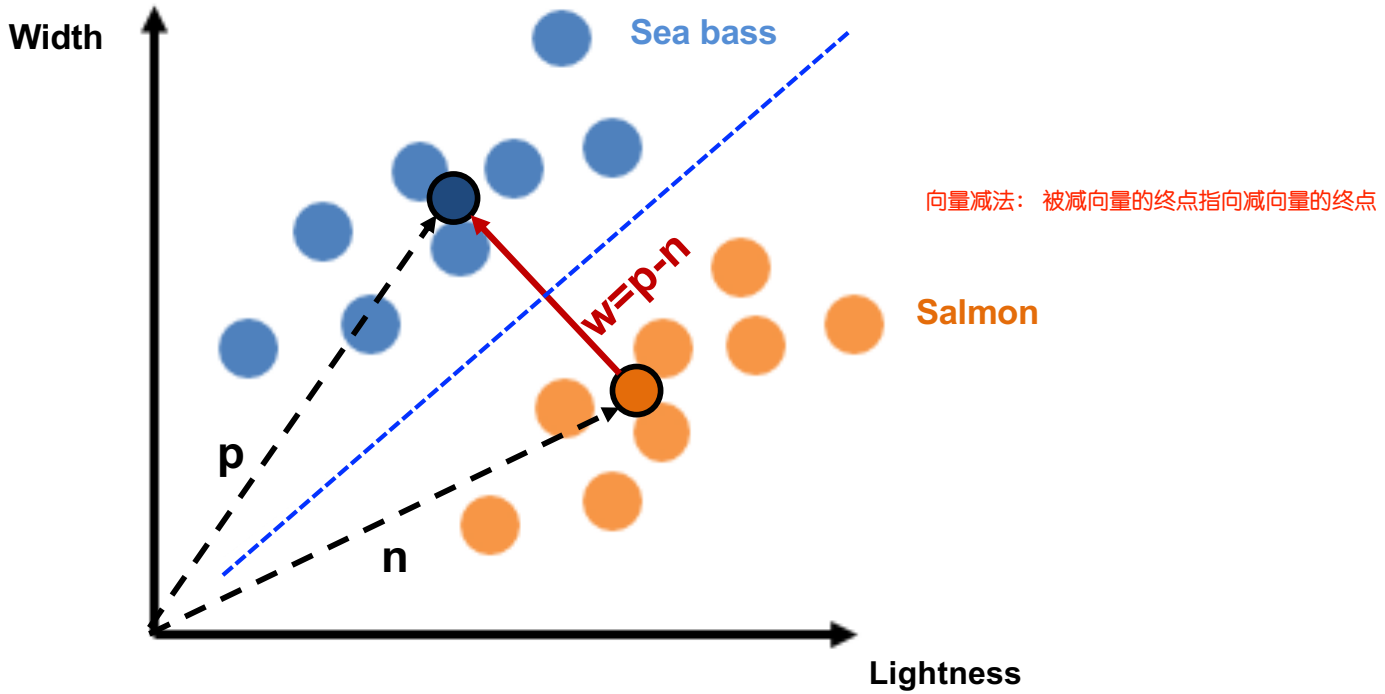
Linear classification in two dimensions



- We find the line that separates the two class: $ax_1 + bx_2 + c = 0$
- We define a weight vector $w^T = [a, b]$, $x^T = [x_1, x_2]$
- So, the line can be defined by $x^T w = -c = t$
- w is perpendicular (垂直的) to decision boundary (in direction of positive class)
- t is the decision threshold (if $x^T w > \mathrm{t}$ then x belongs to positive class and if $x^T w < \mathrm{t}$ then x belongs to negative class)

Basic Linear Classifier

The basic linear classifier constructs a decision boundary by half-way intersecting the line between the positive and negative centres of mass.



Basic Linear Classifier

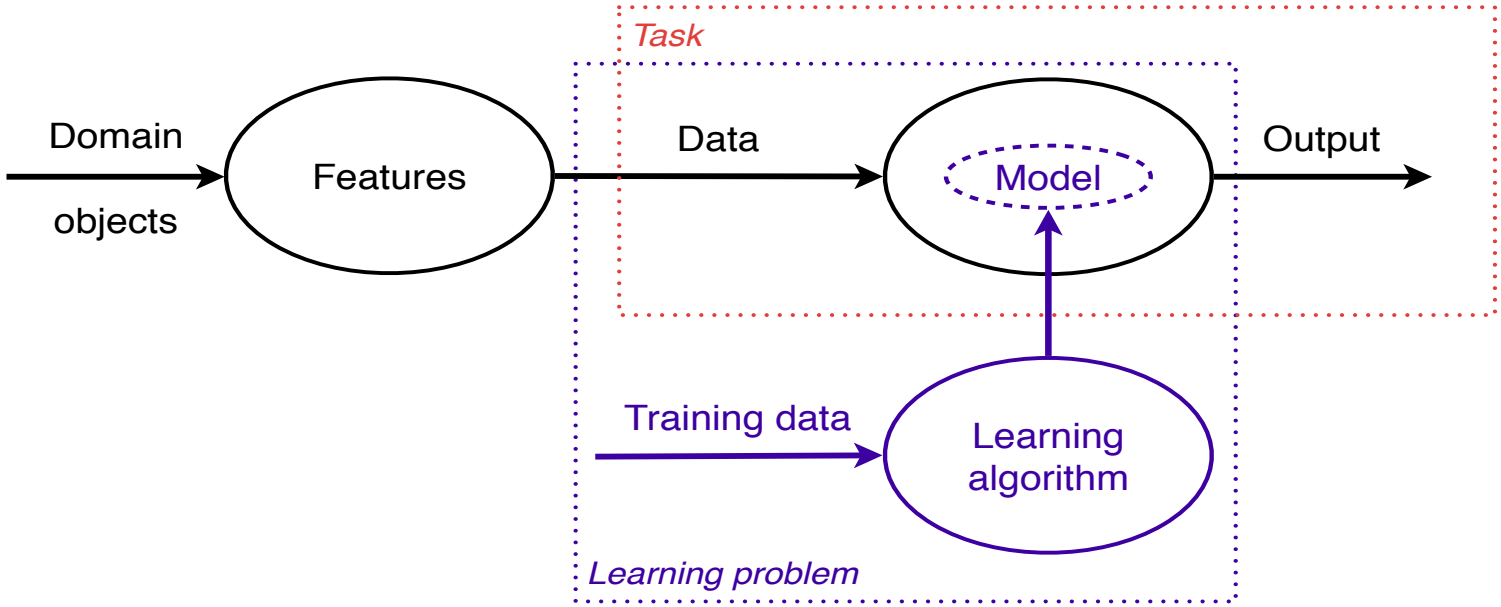
The basic linear classifier is described by the equation $x^T w = t$, and $w = p - n$

As we know, $\frac{p+n}{2}$ is on the decision boundary, so we have:

$$t = (\frac{p+n}{2})^T . (p - n) = \frac{||p||^2 - ||n||^2}{2}$$

Where $||x||$, denotes the length of vector x

How solve a task with machine learning



An overview of how machine learning is used to address a given task. A task (red box) requires an appropriate mapping – a model – from data described by features to outputs. Obtaining such a mapping from training data is what constitutes a learning problem (blue box).

Some terminology

Tasks are addressed by models, whereas learning problems are solved by learning algorithms that produce models.

Some terminology

Machine learning is concerned with using the right features to build the right models that achieve the right tasks.

Some terminology

批量学习（batch learning），一次性批量输入给学习算法，可以被形象的称为填鸭式学习。
在线学习（online learning），按照顺序，循序的学习，不断的去修正模型，进行优化。

Does the algorithm require all training data to be present before the start of learning ? If yes, then it is categorised as **batch learning** (a.k.a. **offline learning**) algorithm.

If, however, it can continue to learn a new data arrives, it is an **online learning** algorithm.

Some terminology

If the model has a fixed 固定的 number of parameters, it is categorised as **parametric**. 参数模型

Otherwise, if the number of parameters grows with the amount of training data it is categorised as **non-parametric**. They do not make any strong assumption about the underlying model and so they are more flexible.

The philosophical problem

演绎：从一般到特殊，从基础原理推演出具体状况，eg从公理推导出定理

Deduction: derive specific consequences from general theories

Induction: derive general theories from specific observations

归纳：从特殊到一般，从具体事实归结出一般性规律，eg从样例中学习

Deduction is well-founded (mathematical logic).

Induction is (philosophically) problematic – induction is useful since it often seems to work – an inductive argument !

Generalisation - the key objective of machine learning

What we are really interested in is generalising from the sample of data in our training set. This can be stated as:

The inductive learning hypothesis

> *Any hypothesis found to approximate the target (true) function well over a sufficiently large set of training examples will also approximate the target function well over other unobserved examples.*

必然结果

A corollary of this is that it is necessary to make some assumptions about the type of target function in a task for an algorithm to go beyond the data, i.e., generalise or learn.

交叉验证Cross-validation

一般需要将样本分成独立的三部分训练集(train set)，验证集(validation set)和测试集(test set)。其中训练集用来估计模型，验证集用来确定网络结构或者控制模型复杂程度的参数，而测试集则检验最终选择最优的模型的性能如何。一个典型的划分是训练集占总样本的50%，而其它各占25%，三部分都是从样本中随机抽取。

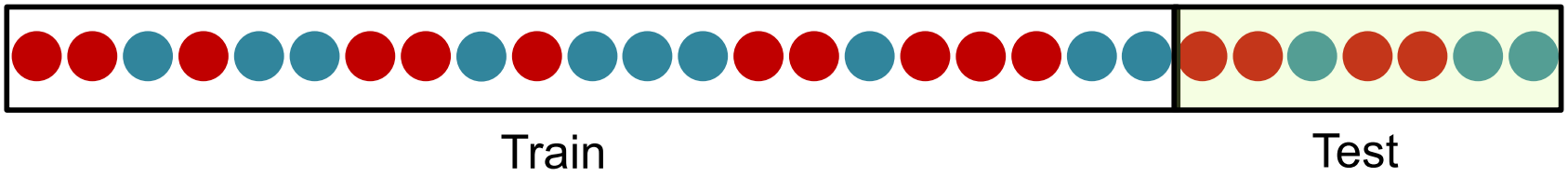
样本少的时候，上面的划分就不合适了。常用的是留少部分做测试集。然后对其余N个样本采用K折交叉验证法。就是将样本打乱，然后均匀分成K份，轮流选择其中K－1份训练，剩余的一份做验证，计算预测误差平方和，最后把K次的预测误差平方和再做平均作为选择最优模型结构的依据。特别的K取N，就是留一法（leave one out）。

training set是用来训练模型或确定模型参数的，如ANN中权值等； validation set是用来做模型选择（model selection），即做模型的最终优化及确定的，如ANN的结构；而 test set则纯粹是为了测试已经训练好的模型的推广能力。当然，test set这并不能保证模型的正确性，他只是说相似的数据用此模型会得出相似的结果。但实际应用中，一般只将数据集分成两类，即training set 和test set，大多数文章并不涉及validation set。

Also known as **out-of-sample testing** is validation technique to assess the results of a model to an independent data set

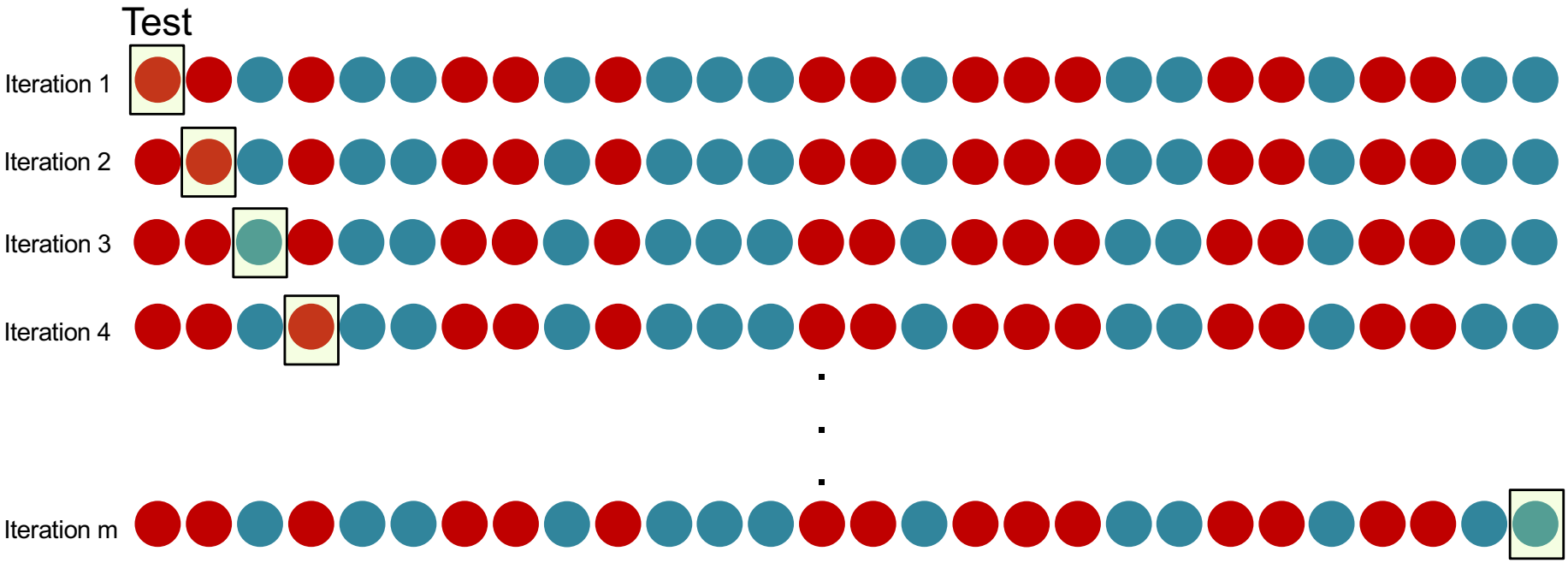
1. Holdout method:

将原始数据随机分为两组，一组做为训练集，一组做为验证集，利用训练集训练分类器，然后利用验证集验证模型，记录最后的分类准确率为此分类器的性能指标。
此种方法的好处的处理简单，只需随机把原始数据分为两组即可，其实严格意义来说Hold-Out Method并不能算是CV，因为这种方法没有达到交叉的思想，由于是随机的将原始数据分组，所以最后验证集分类准确率的高低与原始数据的分组有很大的关系，所以这种方法得到的结果其实并不具有说服性。



Cross-validation

2. Leave-One-Out Cross validation (LOOCV):



如果设原始数据有N个样本，那么LOO-CV就是N-CV，即每个样本单独作为验证集，其余的N-1个样本作为训练集，所以LOO-CV会得到N个模型，用这N个模型最终的验证集的分类准确率的平均数作为此下LOO-CV分类器的性能指标。相比于前面的K-CV，LOO-CV有两个明显的优点：

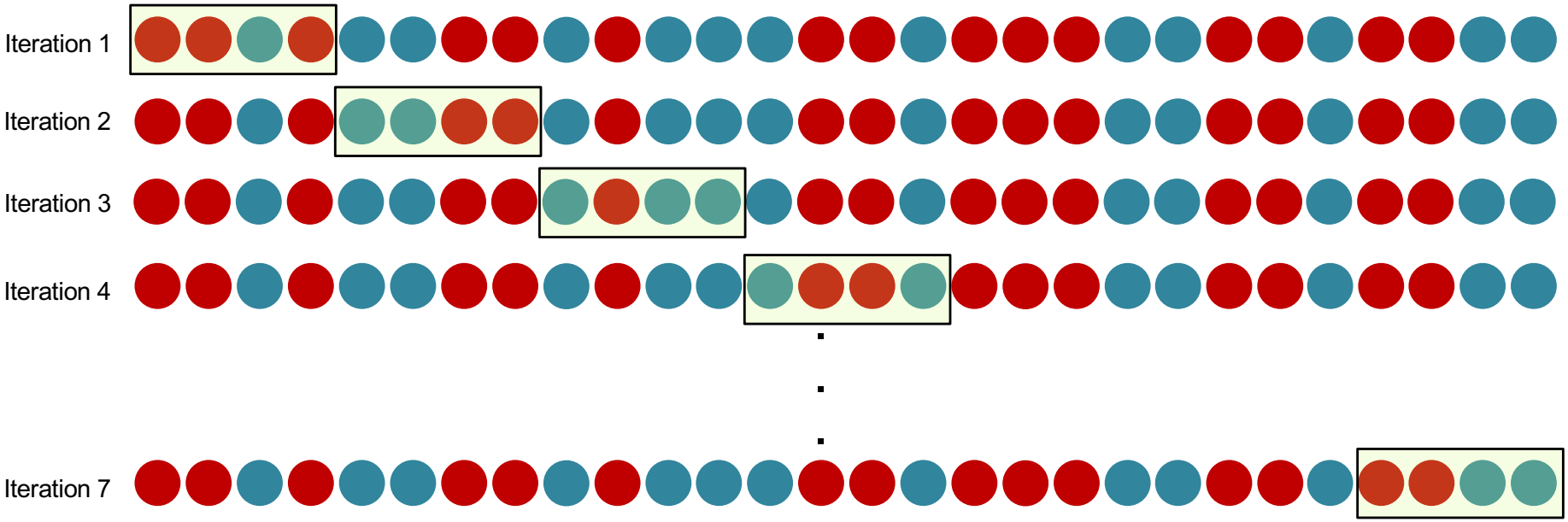
（1）每一回合中几乎所有的样本皆用于训练模型，因此最接近原始样本的分布，这样评估所得的结果比较可靠。

（2）实验过程中没有随机因素会影响实验数据，确保实验过程是可以被复制的。

但LOO-CV的缺点则是计算成本高，因为需要建立的模型数量与原始数据样本数量相同，当原始数据样本数量相当多时，LOO-CV在实作上便有困难几乎就是不显示，除非每次训练分类器得到模型的速度很快，或是可以用并行化计算减少计算所需的时间。

Cross-validation

3. K-fold Cross Validation



将原始数据分成K组（一般是均分），将每个子集数据分别做一次验证集，其余的K-1组子集数据作为训练集，这样会得到K个模型，用这K个模型最终的验证集的分类准确率的平均数作为此K-CV下分类器的性能指标。K一般大于等于2，实际操作时一般从3开始取，只有在原始数据集合数据量小的时候才会尝试取2。K-CV可以有效的避免过学习以及欠学习状态的发生，最后得到的结果也比较具有说服性。
示例中共28个样本， 分成7组，每组4个样本， 共产生7个模型

Cross-validation

There are certain parameters that need to be estimated during learning. We use the data, but NOT the training set, OR the test set. Instead, we use a separate *validation* or *development* set.

为什么验证数据集和测试数据集两者都需要?

因为验证数据集（Validation Set)用来调整模型参数从而选择最优模型，模型本身已经同时知道了输入和输出，所以从验证数据集上得出的误差（Error)会有偏差（Bias)。

但是我们只用测试数据集(Test Set) 去评估模型的表现，并不会去调整优化模型。

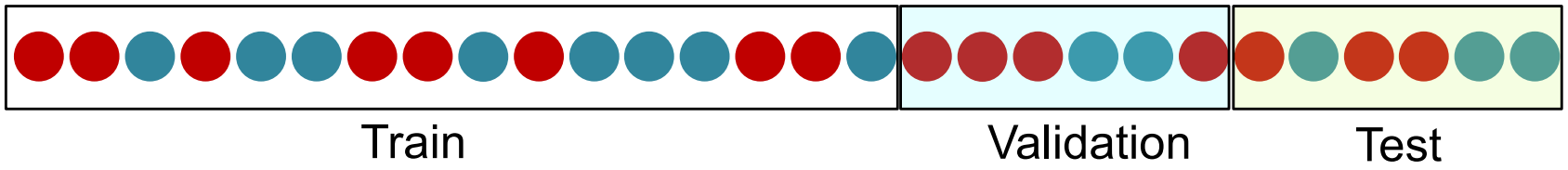
在传统的机器学习中，这三者一般的比例为training/validation/test = 50/25/25, 但是有些时候如果模型不需要很多调整只要拟合就可时，或者training本身就是training+validation (比如cross validation)时，也可以training/test =7/3.

但是在深度学习中，由于数据量本身很大，而且训练神经网络需要的数据很多，可以把更多的数据分给training，而相应减少validation和test。

Cross-validation

超参数调试

Validation set: To make the hyperparameter tuning and model selection independent from the test set, we define another set within the train set



train
训练数据。拟合模型，用这部分数据来建立模型。

是一些我们已经知道输入和输出的数据集训练机器去学习，通过拟合去寻找模型的初始参数。例如在神经网络（Neural Networks)中， 我们用训练数据集和反向传播算法（Backpropagation）去每个神经元找到最优的比重（Weights)。

validation
验证数据。train建了一个模型，但是模型的效果仅体现了训练数据，但不一定适合同类的其他数据，所以建模前数据分成两部分，一部分为训练数据，一部分为验证数据（两部分数据的比例大致为7:3，这取决于你验证的方法）。另外，你也可能训练多个模型，但不知哪个模型性能更佳，这时可以将验证数据输入不同模型进行比较。

是一些我们已经知道输入和输出的数据集，通过让机器学习去优化调整模型的参数，在神经网络中， 我们用验证数据集去寻找最优的网络深度（number of hidden layers)，或者决定反向传播算法的停止点；在普通的机器学习中常用的交叉验证（Cross Validation) 就是把训练数据集本身再细分成不同的验证数据集去训练模型。

test
测试数据。跟前两者的最大区别在于：train和validation数据均是同一对象的数据，但是测试，我们就需要用跨对象的数据来验证模型的稳定性。

用户测试模型表现的数据集，根据误差（一般为预测输出与实际输出的不同）来判断一个模型的好坏。

Data Types

In Machine Learning world, in general two types of data is defined:

- **Numerical**: Anything represented by numbers (e.g., integer , floating point)
- **Categorical**: everything that is not numerical (e.g. discrete labeled groups)

In general, for machine learning algorithms, data has to be represented in numeric form

Data Types

Another taxonomy (分类学，分类系统) of data types:

1. **Irrelevant**: it might be represented with strings or numbers but has no relationship with the outcome (e.g. participants name or code)
2. **Nominal**: discrete values with no numerical relationship between different categories (e.g. animal types, colors, nationality)
3. **Binary**: discrete data with only two possibilities (e.g cancerous vs. non-cancerous)

Data Types

序数的

4. **Ordinal**: discrete integers that can be ranked, but the relative distance between any two number can not be defined (e.g. students rank based on GPA)
5. **Count**: discrete whole numbers without any negatives
6. **Time**: a cyclical, repeating continuous form of data (e.g days, weeks)
7. **Interval**: data that the we can measure the distance between different values. (e.g temperature, income)

Binary Classification task

In a binary classification (or binomial classification) task, we always want to classify the data of a given set into two groups. We usually define one of the classes as positive and one as negative.

- Sometimes the classes are equally important (e.g. recognition of dog vs cat in image classification
- Sometimes misclassification in one of the classes is more costly than misclassification in the other class (e.g. predicting that someone has cancer while (s)he doesn't have vs predicting that someone doesn't have cancer while (s)he has) therefore we may prefer to have better classification in one class in the cost of more errors in the other class

Evaluation of error

If we have a binary classification, then we have two classes of $y \in \{0,1\}$, where we call the class $y = 1$, *positive class* and $y = 0$, *negative class.*

Some evaluation metrics:

- **True positive:** number of instances from class one that have been predicted as one
- **True negative:** number of instances from class zero that have been predicted as zero
- **False positive:** number of instances from class zero that have been predicted as one
- **False negative:** number of instances from class one that have been predicted as zero

Contingency table

For two-class prediction case:

Actual Class	Predicted Class	
	Positive	Negative
Positive	True Positive (TP)	False Negative (FN)
Negative	False Positive (FP)	True Negative (TN)

混淆矩阵

This is also called *confusion matrix*

Classification Accuracy

Classification Accuracy on a sample of labelled pairs $(x, c(x))$ given a learned classification model that predicts, for each instance x, a class value $\hat{c}(x)$:

$$acc = \frac{1}{|Test|} \sum_{x \in Test} I[\hat{c}(x) = c(x)]$$

ACC = (TP + TN) / (TP + TN + FP + FN)

where $Test$ is a test set and $I[]$ is the indicator function which is 1 iff its argument evaluates to true, and 0 otherwise.

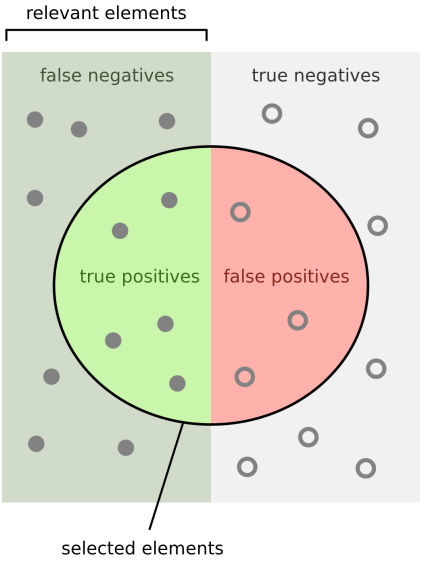
$$Classification\ Error\ is\ = 1 - acc.$$

Other evaluation metrics

Precision/correctness

- is the number of relevant objects classified correctly divided by the total number of relevant objects classified

$$Precision = \frac{TP}{TP + FP}$$



https://en.wikipedia.org/wiki/Precision_and_recall

Recall/sensitivity/completeness/true positive rate (TPR)

- is the number of relevant objects classified correctly divided by total number of relevant/correct objects

$$Recall = \frac{TP}{TP + FN}$$

Other evaluation metrics

F_1 score: a measure of accuracy, which is the harmonic mean of precision and recall and is defined as:

调和平均值

$$F_1 = 2.\frac{precision.recall}{precision + recall}$$

This measure gives equal importance to precision and recall which is sometime undesirable; so we have to decide which metric to use depending on the task and what's important for the task.

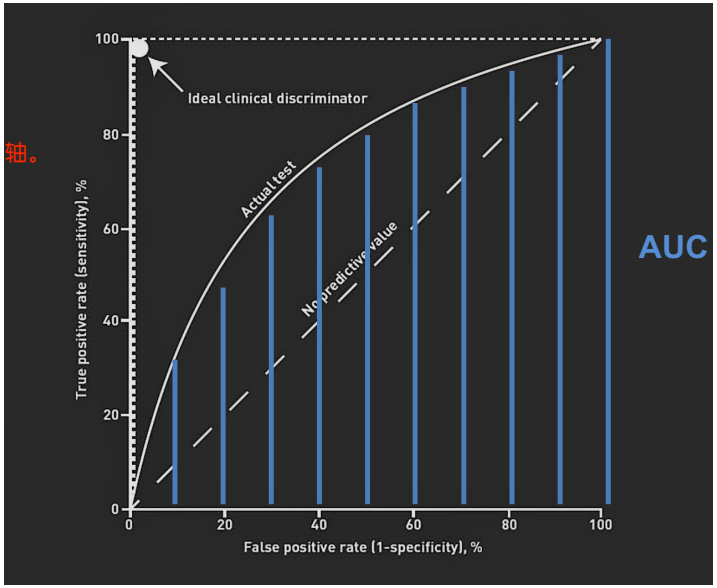
Other evaluation metrics

“现实中样本在不同类别上的不均衡分布(class distribution imbalance problem)。使得accuracy这样的传统的度量标准不能恰当的反应分类器的performance。举个例子：测试样本中有A类样本90个，B 类样本10个。分类器C1把所有的测试样本都分成了A类，分类器C2把A类的90个样本分对了70个，B类的10个样本分对了5个。则C1的分类精度为 90%，C2的分类精度为75%。但是，显然C2更有用些。另外，在一些分类问题中犯不同的错误代价是不同的(cost sensitive learning)。这样，默认0.5为分类阈值的传统做法也显得不恰当了。”
正是因为这样的原因，ROC曲线应运而生。它能够很好的描述分类器对于不均衡分布的样本的分类性能。

AUC-ROC curve: Area Under the Curve (AUC) – Receiver Operating Characteristics (ROC) curve is one of the most important evaluation metric for performance of classification models. This metric evaluates the model at different threshold settings and can inform us on the capability of the model in distinguishing between classes.

受试者工作特征曲线

- $TPR = \frac{TP}{TP+FN}$ (recall)
  ROC空间将伪阳性率（FPR）定义为 X 轴，真阳性率（TPR）定义为 Y 轴。
- $FPR = \frac{FP}{FP+TN}$
  TPR：在所有实际为阳性的样本中，被正确地判断为阳性之比率。
  FPR：在所有实际为阴性的样本中，被错误地判断为阳性之比率
- A good model has AUC close to 1
- A very poor model has AUC close to 0
- $AUC = 0.5$ means no class separation



Missing Value: An issue to consider

Missing Values

- In practice it rarely happens that the data is complete and homogenous.
- Why data is incomplete:
  - Human errors
  - Sensor errors
  - Software bugs
  - Faulty preprocessing
  - …

Missing Values

How to handle missing values (common approaches):

- Deleting samples with missing values
- Replacing the missing value with some statistics from the data (mean, median, …)
- Assigning a unique category
- Predicting the missing values
- Using algorithms that support missing values

Missing Values

Deleting samples with missing values:

- Pros:
  - A robust and probably more accurate model
- Cons:
  - Loss of information and data
  - Works poorly if the percentage of missing values is high

Missing Values

Replacing the missing value with mean/median/mode:

- Pros:
  - When the data size is small, it is better than deleting
  - It can prevent data loss
- Cons:
  - Imputing the approximations adds bias to the model (it reduces the variance of the variable)
  - Works poorly compared to other methods

Missing Values

If categorical, assigning a unique category or the most frequent category:

- Pros:
  - Works well with small datasets and easy to implement
  - No loss of data
- Cons:
  - Works only for categorical features
  - Adding another feature (e.g. a new unique category) to the model may result higher variance in the model
  - Adding the most frequent category can increase the bias in the model

Missing Values

Predicting the missing values:

- Pros:
  - Imputing the missing variable is an improvement as long as the bias from it is smaller than the omitted variable bias
  - Yields unbiased estimates of the model parameters
- Cons:
  - Bias also arises when an incomplete conditioning set is used for a categorical variable
  - Considered only as a proxy for the true values

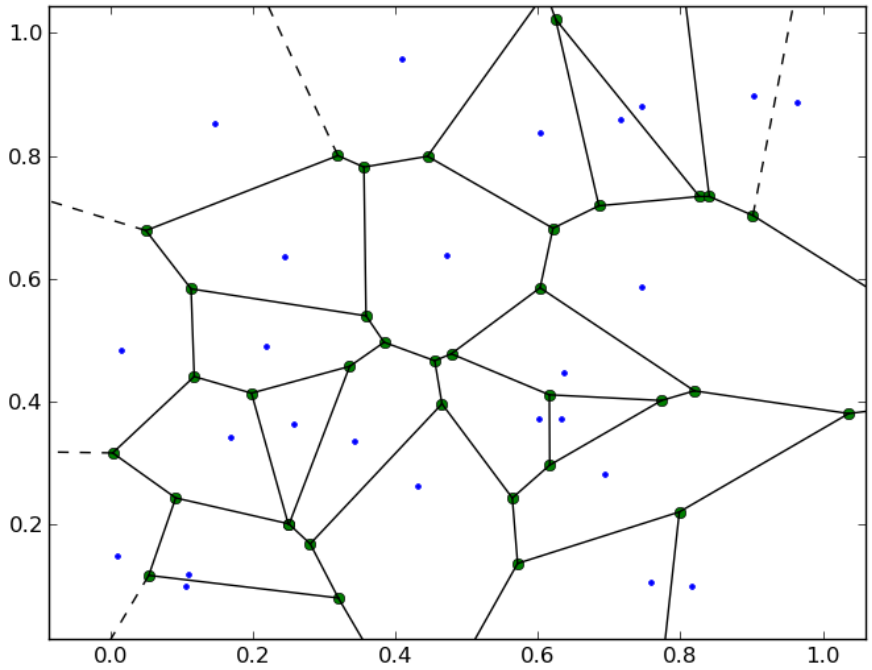
Missing Values

Using algorithms that support missing values:

- Pros:
  - Does not require creation of a predictive model
  - Correlation of the data is neglected 忽视相关性
- Cons:
  - Some of these algorithms are very time-consuming and it can be critical 挑剔的 in data mining where large databases are being extracted

Nearest Neighbor Algorithm for Classification

Nearest Neighbour



Nearest Neighbour is a regression or classification algorithm that predicts whatever is the output value of the nearest data point to some query.

To find the nearest data point, we have to find the *distance* between the query and other points. So we have to decide how to define the *distance*.

Minkowski distance

闵可夫斯基距离

Minkowski distance If $\mathcal{X} \rightarrow \mathbb{R}^d, x, y \in \chi$, the Minkowski distance of order $p > 0$ is defined as:

$$Dis_p(x,y) = (\sum_{j=1}^{d} |x_j - y_j|^p)^{1/p} = ||\mathrm{x} - \mathrm{y}||_p$$

Where $||z||_p = (\sum_{j=1}^{d} |z_j|^p)^{1/p}$ is the $p - norm$ (sometimes denoted L_p norm) of the vector z.

Minkowski distance

- The *2-norm* refers to the familiar *Euclidean distance*

$$Dis_2(x,y) = \sqrt{\sum_{j=1}^{n}(x_j - y_j)^2} = \sqrt{(x-y)^T(x-y)}$$

- The *1-norm* denotes *Manhattan distance*, also called *cityblock* distance:

$$Dis_1(x,y) = \sum_{j=1}^{n}|x_j - y_j|$$

Minkowski distance

- If we now let p grow larger, the distance will be more and more dominated by the largest coordinate-wise distance, from which we can infer that $Dis_\infty = max_j|x_j - y_j|$; this is also called *Chebyshev distance*.

切比雪夫距离

- You will sometimes see references to the *0-norm* (or L_0 norm) which counts the number of non-zero elements in a vector. The corresponding distance then counts the number of positions in which vectors x and y differ. This is not strictly a Minkowski distance; however, we can define it as:

$$Dis_0(x,y) = \sum_{j=1}^{d}(x_j - y_j)^0 = \sum_{j=1}^{d} I[x_j \neq y_j]$$

under the understanding that $x^0 = 0$ for $x = 0$ and 1 otherwise.

Minkowski distance

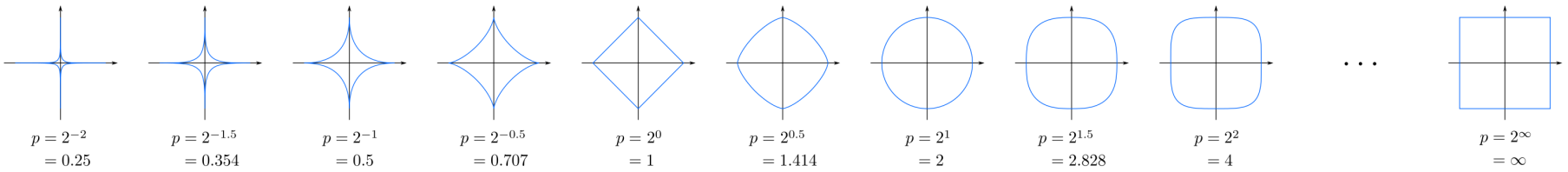
Sometimes the data is not naturally in $\mathbb{R}^d$, but if we can turn it into Boolean features, or character sequences, we can still apply distance measures. For example:

汉明距离
表示两个等长字符串在对应位置上不同字符的数目

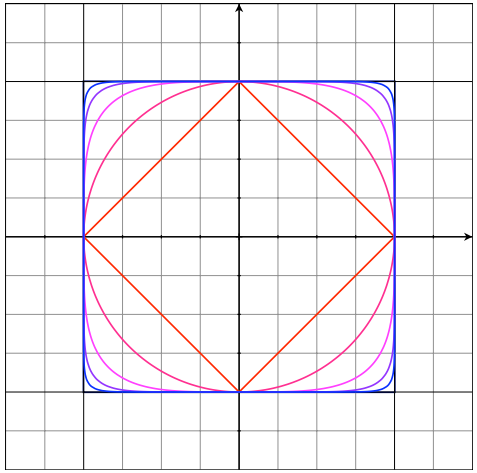
- If x and y are binary strings, this is also called the *Hamming distance*. Alternatively, we can see the Hamming distance as the number of bits that need to be flipped to change x into y.
- For non-binary strings of unequal length this can be generalised to the notion of *edit distance* or *Levenshtein distance*.

编辑距离，至少需要多少次的处理才能将一个字符串变成另一个字符串

Circles and ellipses



Unite circles with different order-p Minkowski distance



- Notice that for points on the coordinate axes all distances agree
- If we require a rotation 旋转不变 invariant distance metric, then Euclidean distance is our only choice

Distance metric

Distance metric Given an instance space $\mathcal{X}$, a distance metric is a function $Dis : \mathcal{X} \times \mathcal{X} \rightarrow [0, \infty)$ such that for any $x, y, z \in \mathcal{X}$:

- distances between a point and itself are zero: $Dis(x, x) = 0$
- all other distances are larger than zero: if $x \neq y$ then $Dis(x, y) > 0$
- distances are symmetric: $Dis(y, x) = Dis(x, y)$
- detours can not shorten the distance (triangle inequality): 绕行

$$Dis(x, z) \leq Dis(x, y) + Dis(y, z)$$

  - It can be shown that triangle inequality does not hold for $p < 1$

If the second condition is weakened to a non-strict inequality – i.e., $Dis(x, y)$ may be zero even if $x \neq y$ – the function Dis is called a *pseudo-metric*. 伪度量

Means and distances

The arithmetic mean minimises squared Euclidean distance *The arithmetic mean μ of a set of data points D in a Euclidean space is the unique point that minimises the sum of squared Euclidean distances to those data points.*

Proof. We will show that $\arg\min_y \sum_{x \in D} ||x - y||^2 = \mu$, where $||.||$ denotes the *2-norm*. We find this minimum by taking the gradient (the vector of partial derivatives with respect to y) of the sum and setting it to the zero vector:

$$\nabla_y \sum_{x \in D} ||x - y||^2 = -2 \sum_{x \in D} (x - y) = -2 \sum_{x \in D} x + 2|D|y = 0$$

From which we derive $y = \frac{1}{|D|} \sum_{x \in D} x = \mu$

Means and distances

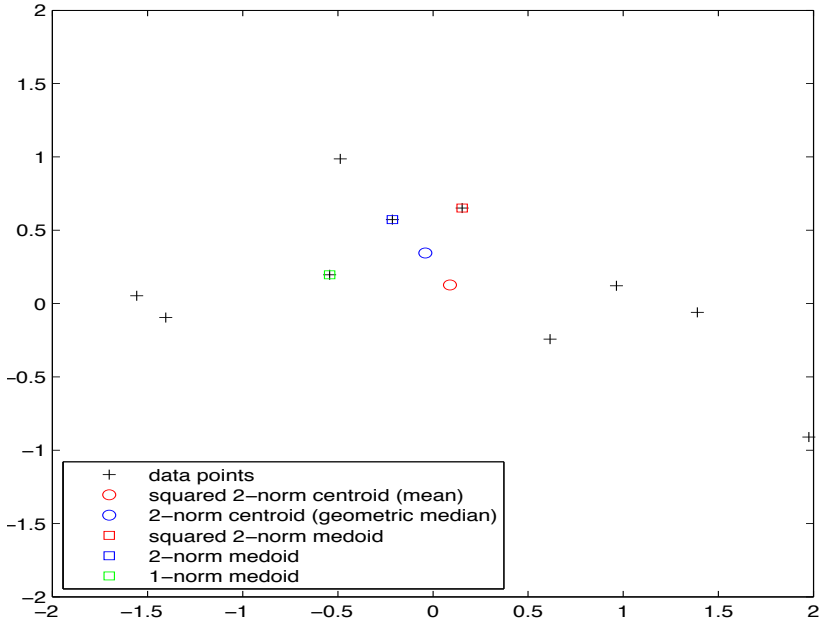
- Notice that minimising the *sum* of squared Euclidean distances of a given set of points is the same as minimising the *average* squared Euclidean distance.
- You may wonder what happens if we drop the square here: wouldn't it be more natural to take the point that minimises total Euclidean distance as exemplar?
- This point is known as the *geometric median*, as for univariate data it corresponds to the *median* or 'middle value' of a set of numbers. However, for multivariate data there is no closed-form expression for the geometric median, which needs to be calculated by successive approximation.

几何中位数

Means and distances

- In certain situations, it makes sense to restrict an exemplar (范例) to be one of the given data points. In that case, we speak of a *medoid* (中心点), to distinguish it from a *centroid* which is an exemplar that doesn't have to occur in the data.
- Finding a medoid requires us to calculate, for each data point, the total distance to all other data points, in order to choose the point that minimises it. Regardless of the distance metric used, this is an $O(n^2)$ operation for n points.
- So, for medoids there is no computational reason to prefer one distance metric over another.
- There may be more than one medoid.

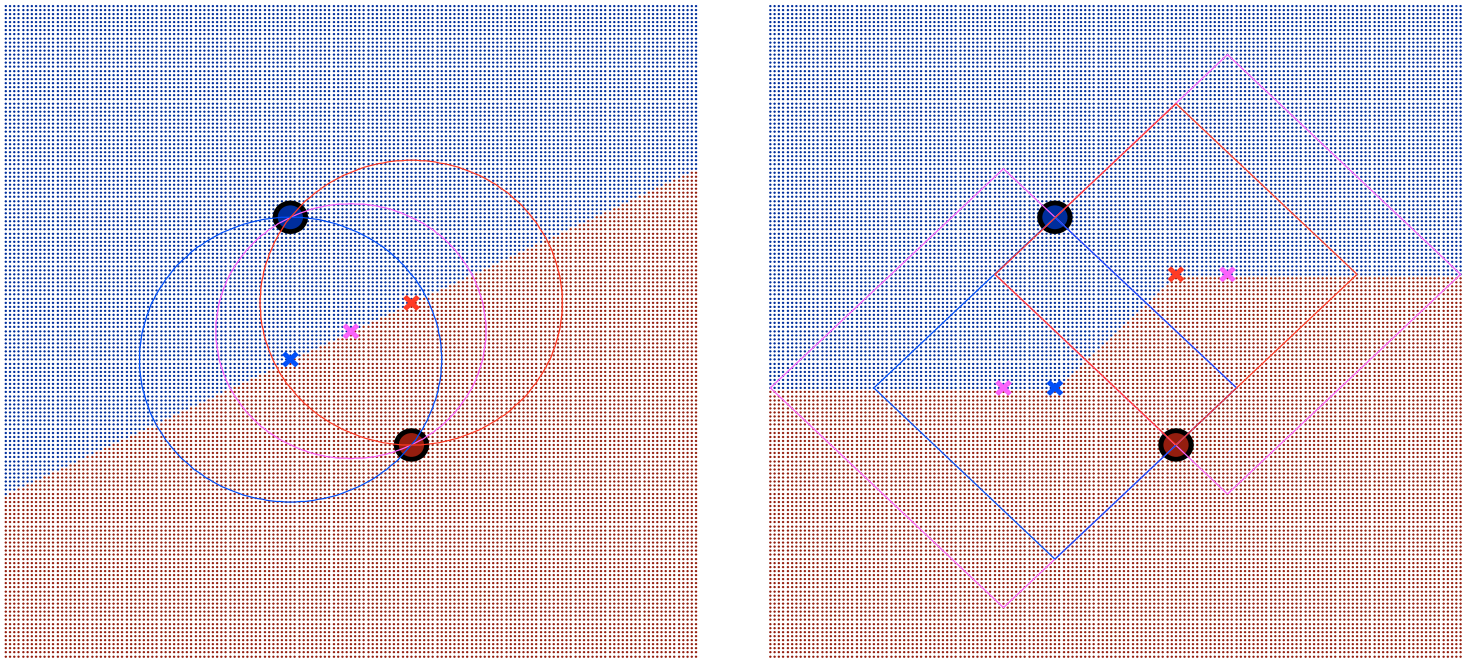
Centroids and medoids



A small data set of 10 points, with circles indicating centroids and squares indicating medoids (the latter must be data points), for different distance metrics. Notice how the outlier on the bottom-right 'pulls' the mean away from the geometric median; as a result, the corresponding medoid changes as well.

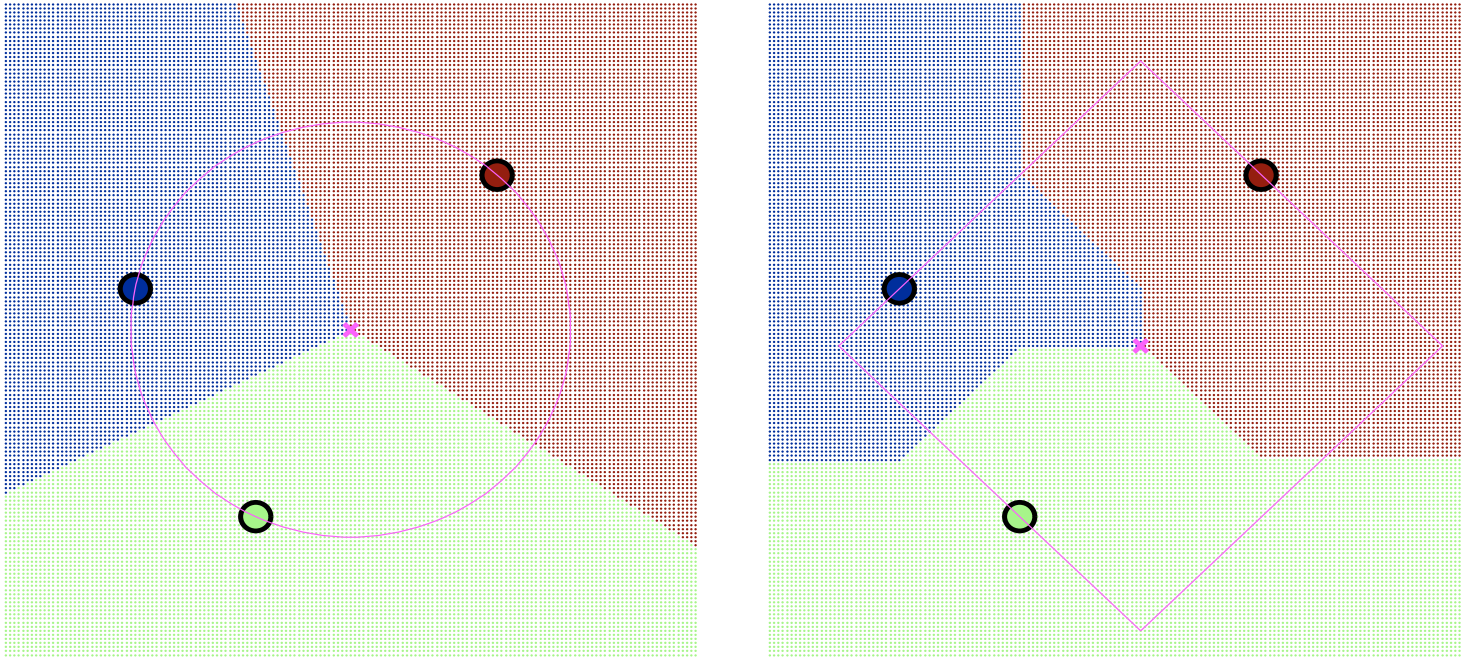
离群值

Two-exemplar decision boundaries



(left) For two exemplars the nearest-exemplar decision rule with Euclidean distance results in a linear decision boundary coinciding with the perpendicular bisector of the line connecting the two exemplars. (right) Using Manhattan distance the circles are replaced by diamonds.

Three-exemplar decision boundaries



(left) Decision regions defined by the 2-norm nearest-exemplar decision rule for three exemplars. (right) With Manhattan distance the decision regions become non-convex.

Distance-based models

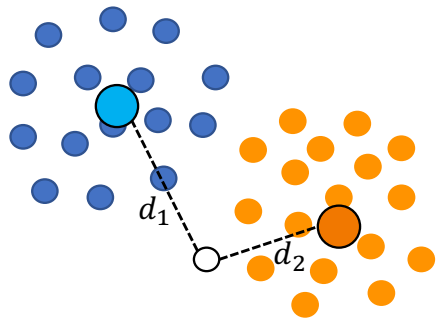
To summarise, the main ingredients of distance-based models are:

- distance metrics, which can be Euclidean, Manhattan, Minkowski or Mahalanobis, among many others;
- exemplars: centroids that find a centre of mass according to a chosen distance metric, or medoids that find the most centrally located data point; and

Nearest Centroid Classifier

Nearest Centroid Classifier

- This is a classifier based on minimum distance principle, where the class exemplars are just the centroids (or means)



- Training: for training sample pairs $\{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$ where x_i is the feature vector for sample i and y_i is the class label, class centroids are:

$$\mu_k = \frac{1}{|C_k|} \sum_{j \in C_k} x_j$$

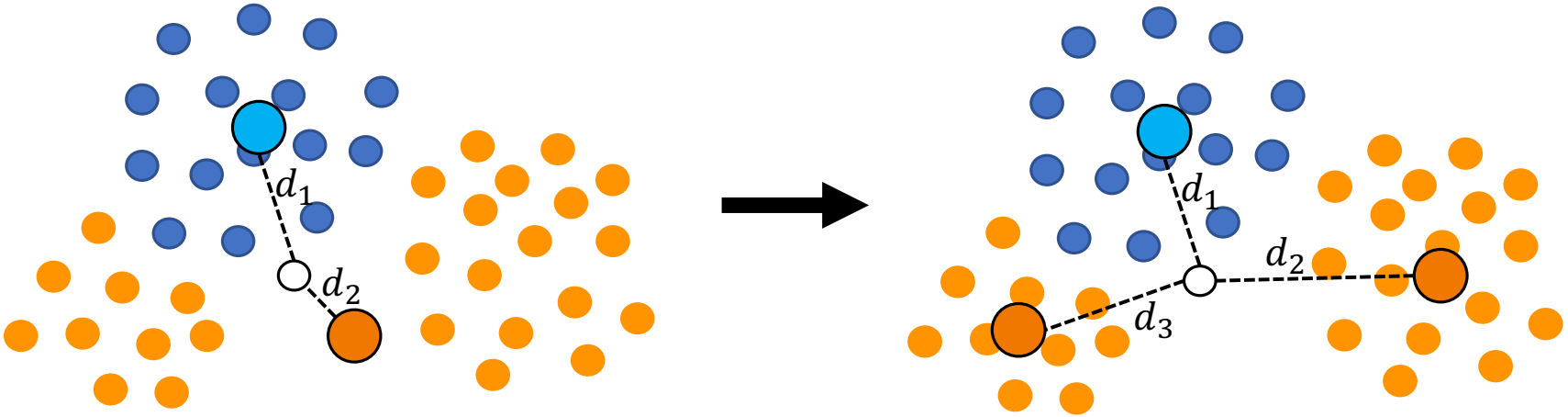
- Test: a new unknown object with feature vector x is classified as class i if it is much closer to the mean vector of class k than to any other class mean vector

Basic Linear Classifier & Nearest Centroid Classifier

- The basic linear classifier is distance-based.
- An alternative, distance-based way to classify instances without direct reference to a decision boundary is by the following decision rule: if x is nearest to $\mu^{\oplus}$ then classify it as positive, otherwise as negative; or equivalently, classify an instance to the class of the nearest exemplar.
- If we use Euclidean distance as our closeness measure, simple geometry tells us we get exactly the same decision boundary.
- So the basic linear classifier can be interpreted from a distance-based perspective as constructing exemplars that minimise squared Euclidean distance within each class, and then applying a nearest-exemplar decision rule.

Nearest Centroid Classifier

- What happens if a class has more than one mode? (similar to the image)
  1. If there is only one centroid per class, then it will perform poorly
  2. If we can somehow find different modes, we can define one centroid per each mode which helps the classifier



(1) (2)

Nearest Centroid Classifier

Advantages:

- Simple
- Fast
- works well when classes are compact (紧实的) and far from each other.

Nearest Centroid Classifier

Disadvantages:

- For complex classes (e.g., Multimodal, non-spherical) may give very poor results
- Can not handle outliers and noisy data well
- Can not handle missing data

Nearest neighbour classification

Nearest neighbour classification

- Related to the simplest form of learning: rote learning or memorization
  - Training instances are searched for instance that most closely resembles (类似于) new or *query* instance
  - The instances themselves represent the knowledge
  - Called: *instance-based*, *memory-based* learning or *case-based* learning; often a form of *local* learning
- The *similarity* or *distance* function defines "learning", i.e., how to go beyond simple memorization
- Intuitive idea — instances "close by", i.e., neighbours or *exemplars*, should be classified similarly
- Instance-based learning is lazy learning
- Methods: *nearest-neighbour*, *k-nearest-neighbour*, …
- Ideas also important for *unsupervised* methods, e.g., clustering (later lectures)

Nearest Neighbour

Stores all training examples $\langle x_j, f(x_j) \rangle$.

Nearest neighbour:

- Given query instance x_q, first locate nearest training example x_n, then estimate $\hat{f}(x_q) \leftarrow f(x_n)$

k-Nearest neighbour:

- Given x_q, take vote among its k nearest neighbours (if discrete-valued target function) (see next slide)
- take mean of f values of k nearest neighbours (if real-valued)

$$\hat{f}(x_q) \leftarrow \frac{\sum_{j=1}^{k} f(x_j)}{k}$$

k-Nearest Neighbour Algorithm

Training algorithm:

- For each training example $\langle x_j, f(x_j) \rangle$, add the example to the list *training _examples*.

Classification algorithm:

- Given a query instance x_q to be classified,
  - Let $x_1, \ldots, x_k$ be the k instances from *training examples* that are *nearest* to x_q by the distance function
  - Return

$$\hat{f}(x_q) \leftarrow \arg\max_{v \in V} \sum_{i=1}^{k} \delta(v, f(x_j))$$

Where $\delta(a, b) = 1$ if $a = b$ and 0 otherwise.

Distance function again

The distance function defines what is learned.
Instance x_j is described by a feature vector (list of attribute-value pairs)

$$x_j = (x_{j1}, \ldots, x_{jd})^T$$

Where x_{jr} denotes the value of the rth attribute/feature of x_j.

Most commonly used distance function is *Euclidean distance* . . .

- distance between two instances x_i and x_j is defined to be

$$Dis(x_i, x_j) = \sqrt{\sum_{r=1}^{d} (x_{jr} - x_{jr})^2}$$

Distance function again

Many other distance functions could be used . . .

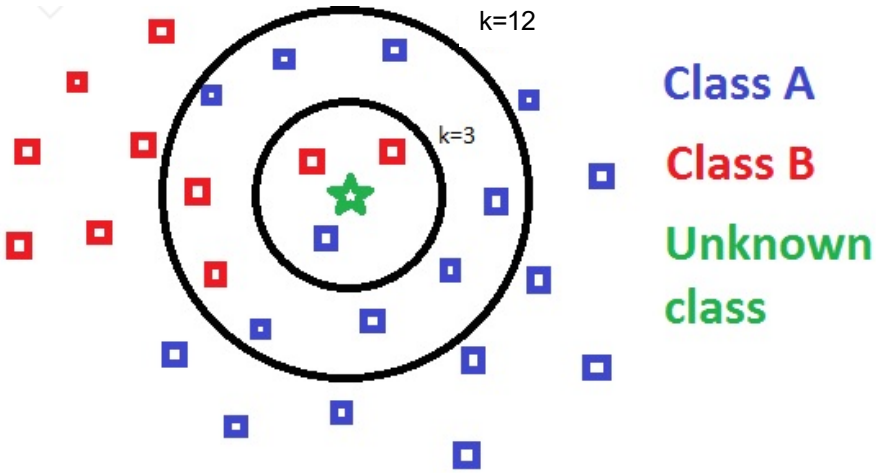
- e.g., *Manhattan* also referred to as *city-block* distance (sum of absolute values of differences between attributes)

$$Dis(x_i, x_j) = \sum_{r=1}^{d} |x_{ir} - x_{jr}|$$

Vector-based formalization – use norm L_1, L_2, …

kNN Example

- What is the predicted class for the green point given the data for?



https://towardsdatascience.com/knn – using – scikit – learn – c6bed765be75

Normalization and other issues

- Different attributes measured on different scales (for example one attribute/feature may have a range of $[0,100]$ and another have a range of $[-1,1]$)
- Need to be *normalized* (why ?)

$$x'_{jr} = \frac{x_{jr} - \min(x_{jr})}{max(x_{jr}) - \min(x_{jr})}$$

where x_{jr} is the actual value of attribute/feature r and x'_{jr} is the normalised value.

- Nominal attributes: distance either 0 or 1

When To Consider Nearest Neighbour

- Instances map to points in $\mathbb{R}^d$
- Less than 20 attributes per instance
    - or number of attributes can be reduced . . .
- Lots of training data
- No requirement for “explanatory” model to be learned

K-Nearest Neighbour

Advantages:

- Statisticians have used k-NN since early 1950s
- Can be very accurate
- Training is very fast
- Can learn complex target functions

K-Nearest Neighbour

knn的基础是距离。维度灾难在使用距离的比较时问题尤甚。
会导致这个问题是因为，当维度增大时，距离某个样本点单位距离内的其他样本点数量的比值会减少，这会导致我们寻找更远的距离才能找到临近的值。
注意，虽然看起来对于knn选择的k个样本点并没有影响，但问题是选择的样本点随着维度的增高，距离该样本是越来越远的，因此没有那么有参考价值了。
这是维度灾难对于knn影响特别大的地方。

另一个问题在于，knn每次需要遍历整个样本，这会导致大量计算。

Disadvantages:

- Slow at query time: basic algorithm scans entire training data to derive a prediction
- "Curse of dimensionality" 维数灾难
- Assumes all attributes are equally important, so easily fooled by irrelevant attributes
  - 改进 Remedy: attribute selection or weights
- Problem of noisy instances:
  - Remedy: remove from data set
  - not easy – how to know which are noisy ?
- Needs homogenous 同类的 feature type and scale
- Finding the optimal number of neighbors (k) can be challenging

Inductive Bias of KNN

The **inductive bias** (a.k.a. learning bias) of a learning algorithm is the set of assumptions that the learner uses to predict outputs given unseen inputs.

What is the inductive bias of KNN ?

- an assumption that the classification of query instance x_q will be most similar to the classification of other instances that are nearby according to the distance function

Nearest-neighbour classifier

- 1NN perfectly separates training data, so low bias but high variance
- By increasing the number of neighbours k we increase bias and decrease variance (what happens when $k = m$? m is the number of observations)
- Easily adapted to real-valued targets, and even to structured objects (nearest-neighbour retrieval). Can also output probabilities when $k > 1$

Distance-Weighted KNN

- Might want to weight nearer neighbours more heavily ...
- Use distance function to construct a weight w_i
- Replace the final line of the classification algorithm by:

$$\hat{f}(x_q) \leftarrow \arg\max_{v \in V} \sum_{i=1}^{k} w_i \delta(v, f(x_j))$$

Where

$$w_i = \frac{1}{Dis(x_q, x_i)^2}$$

$Dis(x_q, x_i)$ is distance between x_q, x_i

Distance-Weighted KNN

For real-valued target functions replace the final line of the algorithm by:

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} w_i f(x_i)}{\sum_{i=1}^{k} w_i}$$

(denominator normalizes contribution of individual weights).

Now we can consider using all the training examples instead of just k:

- using all examples (i.e., when $k = m$ and m is number of training samples) with the rule above is called *Shepard's method*

Evaluation

Lazy learners do not construct an explicit model, so how do we evaluate the output of the learning process ?

- 1-NN – training set error is always zero !
  - each training example is always closest to itself
- k-NN – overfitting may be hard to detect

Solution:

Leave-one-out cross-validation (LOOCV) – leave out each example and predict it given the rest:

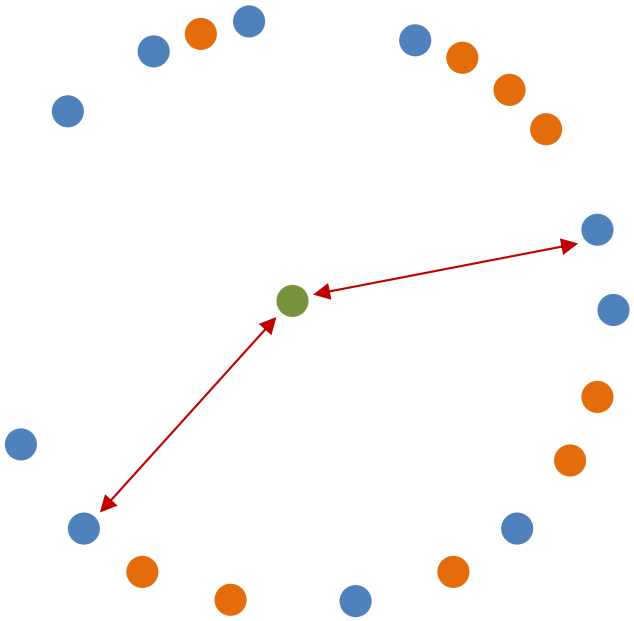
$$(x_1, y_1), (x_2, y_2), \ldots, (x_{i-1}, y_{i-1}), (x_{i+1}, y_{i+1}), \ldots, (x_m, y_m)$$

Error is mean over all predicted examples. Fast – no models to be built !

KNN Computational Time

- KNN uses the training data as exemplars, so using simple search for prediction is $O(n)$!
- There are algorithms to search for neighbours more efficiently with $O(\log n)$ but they do not work very well for above 10 dimensions (more than 10 features/attributes)
- For above 10 dimension, there are some approximate nearest neighbour approaches that can improve computation by orders of magnitude 量级
- In high dimensional space (e.g., above 20 dimensions) even with using such algorithms, the KNN doesn't work well
- In high-dimensional spaces everything is far away from everything and so pairwise distances are uninformative (curse of dimensionality) 信息不足

When is KNN meaningful?

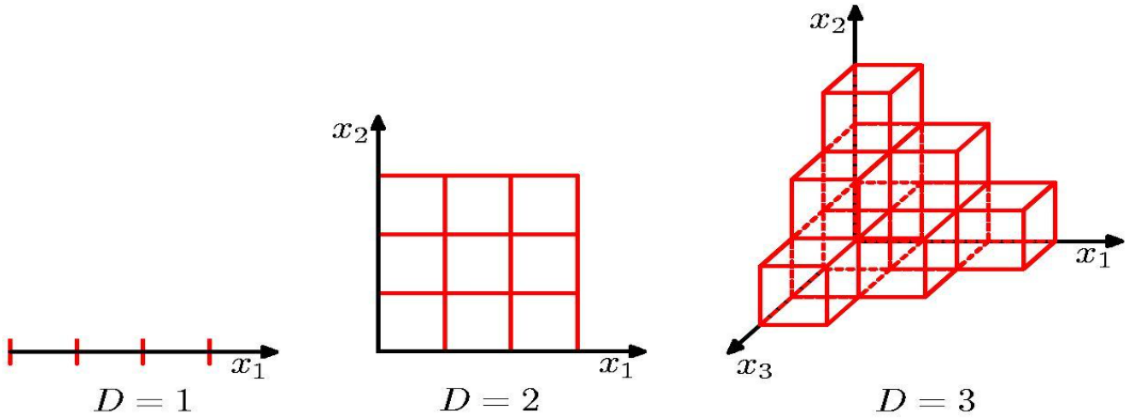


You may think that this is an exceptional example, and this doesn't really happen in practice!!

Curse of Dimensionality

- It can be shown that as dimensions increase the effectiveness of distance metrics decrease and the concept of proximity may not be qualitatively meaningful as all points look equidistant
- This is one symptom of having high dimensional space (curse of dimensionality)
- There are also other problems arising from curse of dimensionality:
  - It becomes polynomially harder to estimate many parameters (e.g., covariances)
  - It becomes more difficult to visualize data
  - Enormous amount of data is needed to train a model

Curse of Dimensionality



- number of “cells” in the instance space grows exponentially in the number of features
- with exponentially many cells we would need exponentially many data points to ensure that each cell is sufficiently populated to make nearest-neighbour predictions reliably

Curse of Dimensionality

Bellman (1960) coined this term in the context of dynamic programming

Imagine instances described by 20 attributes, but only 2 are relevant to target function — "similar" examples will appear "distant".

Curse of dimensionality: nearest neighbour is easily mislead when dealing with high-dimensional x in terms of the number of features – problem of irrelevant attributes

One approach:

- Stretch jth axis by weight z_j, where $z_1, \ldots, z_d$ chosen to minimize prediction error
- Use cross-validation to automatically choose weights $z_1, \ldots, z_d$
- Note setting z_j to zero eliminates this dimension altogether

Curse of Dimensionality

Some ideas to address this for instance-based (nearest-neighbour) learning

- Euclidean distance with weights on attributes

$$Dis(x_q, x_i) = \sqrt{\sum_{r=1}^{d} z_r (x_{qr} - x_{ir})^2}$$

- updating of weights based on nearest neighbour classification error
  - class correct/incorrect: weight increased/decreased
  - can be useful if not all features used in classification

See Moore and Lee (1994) “Efficient Algorithms for Minimizing Cross Validation Error”

Instance-based (nearest-neighbour) learning

Recap – Practical problems of NN scheme:

- Slow (but fast *k-dimensional tree*-based approaches exist)
  - Remedy: removing irrelevant data
- Noise (but KNN copes quite well with noise)
  - Remedy: removing noisy instances
- All attributes deemed equally important
  - Remedy: attribute weighting (or simply selection)

Some refinements of instance-based classifiers

- Edited NN classifiers discard some of the training instances before making predictions (removes datapoint that does not agree with the majority of its k nearest neighbors)
  - Saves memory and speeds up classification
- IB2: incremental (递增的) NN learner that only incorporates (收纳) misclassified instances into the classifier
    - Problem: noisy data gets incorporated
- IB3: store classification performance information with each instance & only use in prediction if above a threshold

Dealing with noise

Use larger values of k (why ?) How to find the “right” k ?

- One way: cross-validation-based k-NN classifier (but slow)
- Different approach: discarding instances that don’t perform well by keeping success records of how well an instance does at prediction (IB3)

kNN Example

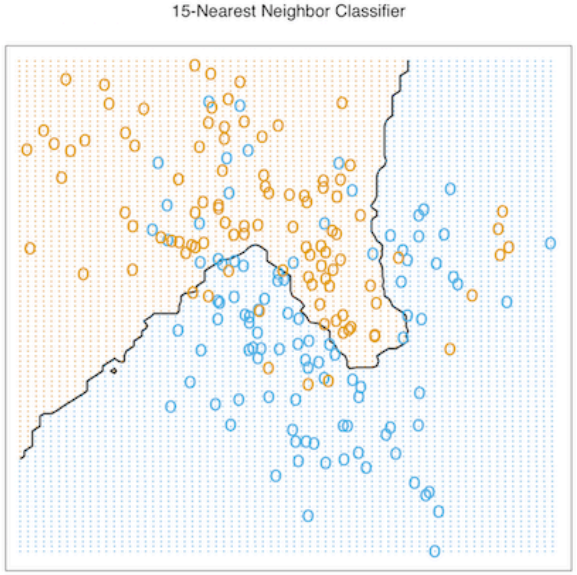


FIGURE 2.2. *The same classification example in two dimensions as in Figure 2.1. The classes are coded as a binary variable* (BLUE $= 0$, ORANGE $= 1$) *and then fit by* 15*-nearest-neighbor averaging as in (2.8). The predicted class is hence chosen by majority vote amongst the* 15*-nearest neighbors.*

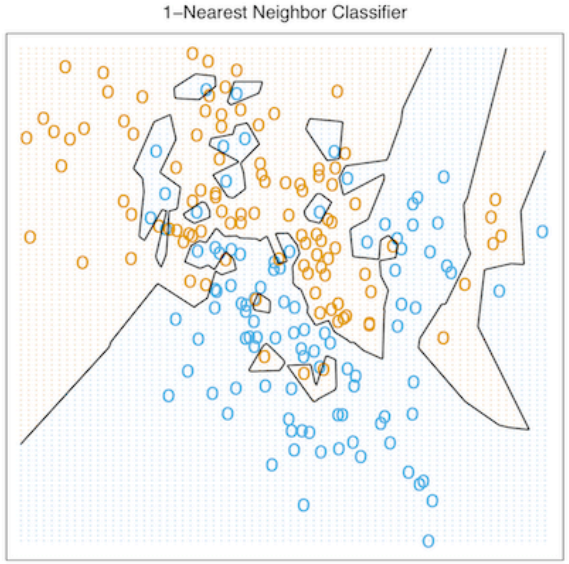


FIGURE 2.3. *The same classification example in two dimensions as in Figure 2.1. The classes are coded as a binary variable* (BLUE $= 0$, ORANGE $= 1$), *and then predicted by* 1*-nearest-neighbor classification.*

kNN Example

- Automated MS-lesion segmentation by KNN
- They have used some manually labeled image as the training set
- They used 4 features: Intensity and voxel locations (x,y,z coordinates)

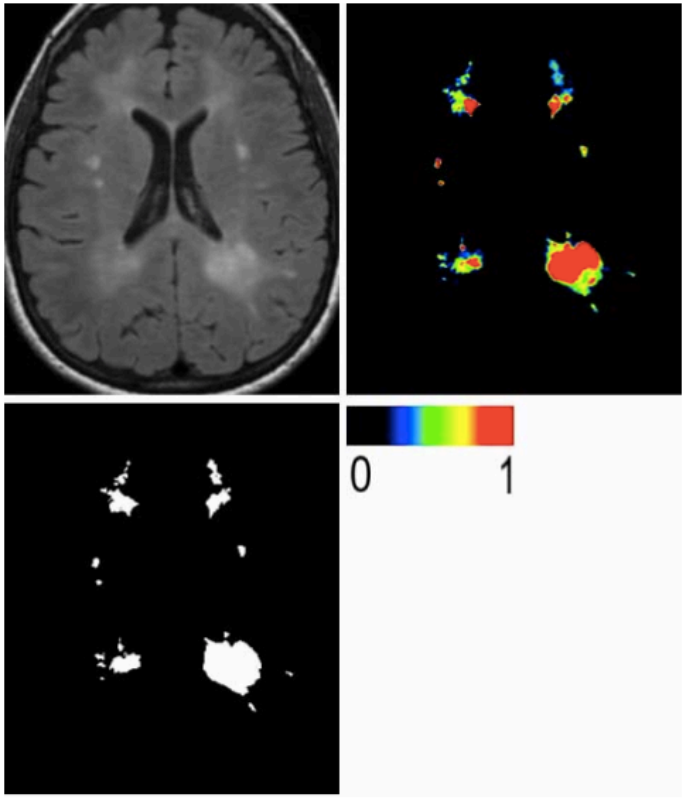


Figure 1 MS-lesion segmentation results. Top left: FLAIR image; top right: probabilistic segmentation, showing probability of lesion per voxel (see color bar); down left: binary segmentation, derived from probabilistic segmentation with threshold 0.4.

- Ref: Anbeek et. Al, "Automated MS-lesion segmentation by K-nearest neighbor classification", MIDAS journal, 2008

Summary

- A framework for classification
- Classification viewed in terms of distance in feature space
- Distance-based learning
- Nearest neighbour classifiers
- Later we will see how to extend by building on these ideas

Acknowledgements

- Material derived from slides for the book
  "Elements of Statistical Learning (2nd Ed.)" by T. Hastie, R. Tibshirani & J. Friedman. Springer (2009) http://statweb.stanford.edu/~tibs/ElemStatLearn/
- Material derived from slides for the book
  "Machine Learning: A Probabilistic Perspective" by P. Murphy MIT Press (2012)
  http://www.cs.ubc.ca/~murphyk/MLbook
- Material derived from slides for the book "Machine Learning" by P. Flach Cambridge University Press (2012) http://cs.bris.ac.uk/~flach/mlbook
- Material derived from slides for the book
  "Bayesian Reasoning and Machine Learning" by D. Barber Cambridge University Press (2012)
  http://www.cs.ucl.ac.uk/staff/d.barber/brml
- Material derived from slides for the book "Machine Learning" by T. Mitchell McGraw-Hill (1997)
  http://www- 2.cs.cmu.edu/~tom/mlbook.html
- Material derived from slides for the course "Machine Learning" by A. Srinivasan BITS Pilani, Goa, India (2016)